Appendix D

Title:
BPI Specification
Draft Version 0.9

1. General

This document describes how to use the Bus-Peripheral-Interface. In the current version, this document is about BPI for FPI bus only.

2. Function

2.1. General concept

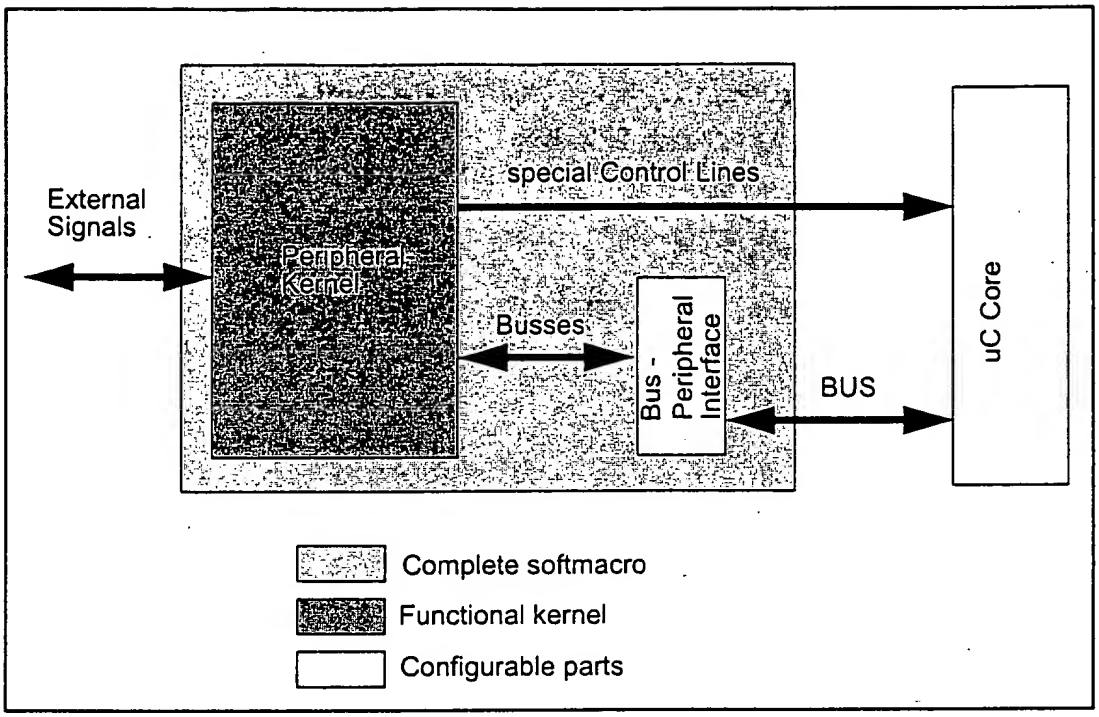
A Bus-Peripheral Interface BPI completely handles the bus protocol of the chip internal interconnect bus; this BPI can therefore be seen as the standardized interface to the FPI-Bus.

On the other side all BPIs adhere to the fundamentals given in "Platform Concept: SMIF Specification" (T. Steinecke, P. Schneider) and considered binding for all platform peripheral.

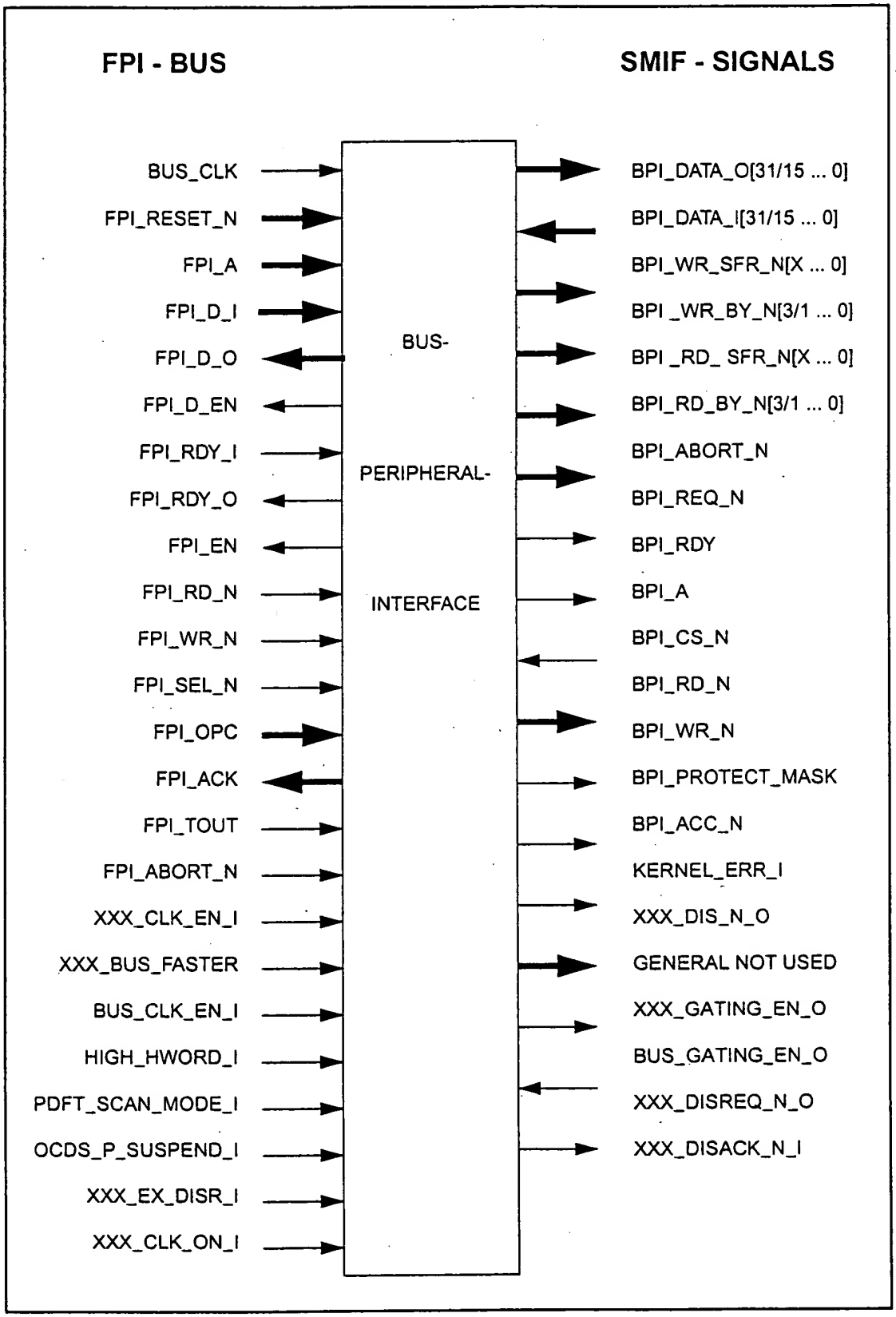
The peripheral's functional code (named "kernel" throughout this document) is thus independent of the used bus protocol.

The main tasks of the BPI are

- Buffering — To define a predictable bus timing some standardisation on fan out and fan in is mandatory
- Decoding — As Kernel SFR's will have different addresses in different uC's, addresses are decoded inside BPI mainly.
- Handshaking — Bus and peripheral may use different clock rates (even dynamically !). Thus some way of synchronization has to be provided.
- Error Handling — Access to undefined locations and unclocked peripherals are functions not regarded legal under normal circumstances; some error notification via the bus may be needed.
- Test Mode — If special test modes are necessary the should be implemented likewise throughout a chip.


External Signals
Peripheral Kernel
special Control Lines
Busses
Bus - Peripheral Interface
BUS
uC Core
Complete softmacro
Functional kernel
Configurable parts

2.2. I/O signals from BPI_INTERFACE



The FPI signals are described in the in the FPI-Specification.

Description of the interface ports to the SMIF:

BUS_CLK	Corresponds to the fpi_clk
fpi_rdy_i	Corresponds to the fpi_rdy signal (fpi_rdy_b) from the FPI
fpi_rdy_o	Ready acknowlege from the BPI. Driven to the signal fpi_rdy_b if enabled with the fpi_en_o signal.
fpi_en_o	Enable signal for the fpi_rdy_o and fpi_ack_o tristate driver
fpi_ack_o	Acknowlege signal from the BPI. Driven to the signal fpi_ack(_b) if enabled with the fpi_en_o signal. **Note: Must be INOUT for synopsys test compiler**
fpi_d_i	Corresponds to the fpi_d from the FPI
fpi_d_o	Driven by the BPI internal data register. Driven to fpi_d(_b) if enabled with the fpi_d_en_o signal.
fpi_d_en_o	Enable signal for the fpi_d_o tristate driver
BPI_DATA_I[31/15...0]	Input data bus, either 32 or 16 bits wide. Source: Peripheral kernel port kernel_data_o
BPI_DATA_O[31/15...0]	Output data bus, either 32 or 16 bits wide. Destination: Peripheral kernel port kernel_data_i
BPI _RD_SFR(x...0)	Dedicated read signals, one each for each SFR.
BPI_RD_BY_N[3...0] (optional)	Selection which bytes have to be read
BPI _WR_ SFR(x...0)	Dedicated write signals, one each for each SFR.
BPI_WR_BY_N[3...0]	Selection which bytes have to be written
BPI_PROTECT_MASK [31/15...0] (optional)	Mask for bit-protection. ‚0' means that the BIT must not be changed ‚1' means that the BIT must be changed in the peripheral.
BPI_ABORT_N (optional)	This signal is directly mapped to the FPI-BUS-signal FPI_ABORT_N. Each peripheral can abort any access in this way. **In the current version this signal is not used by the interface!! This is wrong and will be resolved in the next version**
BPI_REQ_N (optional)	Indicates a valid access if dynamic waitstate insertion is required (handshake_c=1). This signal is reset by BPI_RDY.

BPI_RDY (optional)	Ready indication from the kernel after a dynamic waitstate insertion. While this signal is driven ‚0', it is not possible to request a new access with the signal BPI_REQ_N. Exactly one waitstate is inserted if BPI_RDY is directly bridged to BPI_REQ_N in the kernel
BPI_A (optional)	Synchronized address bits (latched FPI_A) for the RAM-Interface. Only valid if a RAM-Interface is configured
BPI_CS_N (optional)	Chip select signal for the RAM-Interface
BPI_RD_N (optional)	Read signal for the RAM-Interface
BPI_WR_N (optional)	Write signal for the RAM-Interface
BPI_ACC_N	This signal indicates to the kernel when a Write or Read access will take place. It is active for the period of one "Master clock" before and during the rising edge of the peripheral clock. With this signal, multiple reads or writes to a peripheral register can be prevented. A normal read cycle doesn't need this signal, but if a register performes destructive read accesses this signal must be considered.
XXX_BUS_FASTER	Notifies the BPI module of a peripheral that the bus clock rate is currently higher than the peripheral clock rate. This speed indication is needed to perform zero Waitstate write accesses to the peripheral's internal registers whenever the Bus clock rate is equal to or slower than the peripheral clock rate. Connecting XXX_BUS_FASTER to static '1' will insert exactly one waitstate into each access.
KERNEL_ERR	Error signal from the Peripheral Core to the BPI module signalling an error condition during a not allowed RD/WR access from BPI module to the peripheral core. Must be connected to ‚0' if not used. **In the current version this signal needs to be driven before an access will be performed. This is not possible if the peripheral clock is turned off !**
OCDS_P_SUSPEND	Notifies the peripheral to stop the peripheral clock for debugging purposes.
xxx_ex_disr	Requests the peripheral to stop the peripheral clock.
Bus_clk_en	Enable signal for the BPI component clock domain. Is used for clock gating.
xxx_clk_en	Enable signal for the peripheral kernel clock domain. Is used for clock gating.
Bus_gating_en	Enable signal for clock gating of the peripheral clock domain.

xxx_gating_en	Enable signal for clock gating of the peripheral clock domain.
xxx_clk_on	Special function pin for RTC. Must be connected to ‚1' if not used!
xxx_dis_n_o	Special function pin for RTC. Leave open if not used.
xxx_disreq_n	Disable request signal to the kernel to switch off the peripheral clock.
xxx_disack_n	Disable acknowledge signal from the peripheral kernel to allow switching off of the peripheral clock. Bridge this signal to xxx_disreq_n if no specific function is implemented in the kernel
high_hword	**This signal is only considered during write accesses if the fpi data bus width is 32 bits and the kernel data bus width is 16 bits.** High_hword = „0" means that the lower 16 bits of the fpi bus carry valid data and are to be mapped to the kernel data bus. High_hword = „1" means that the upper 16 bits of the fpi bus carry valid data and are to be mapped to the kernel data bus.

2.3. Architecture of the whole peripheral and testbench

A peripheral is a hierarchical construction (see Figure 2 below).
The peripheral kernel can be given either as a structural or as a rtl description.
The components <project>_bpi and <project>_kernel are contained within the module <project>_syn. (syn means synthesis unit)

Example: p3_wdt_bpi and p3_wdt_kernel comprise p3_wdt_syn.

On this project level, no tristate drivers are included, they are instantiated in module <project>_bus_driver.

Example: p3_wdt_bus_driver.

For the distribution of the master clock to the BPI and its kernel a special clock driver each will be instantiatied inside module <project>_clock_gating.

Example: p3_wdt_clock_gating.

The three modules <project>_bus_driver, <project>_clock_gating and <project> _syn are contained inside <project>

The dedicated testbench (<project>_tb) for this toplevel module consist of the peripheral itself (Example: p3_wdt)and the test units FPI (bus model) and <project>_IOC_top (input output control).

Example: p3_wdt_ioc_top.

The figure below shows the internal hierarchy of the bpi_interface and the other components.

Remark: All VHDL units should have a unique prefix in their name to signal their belonging to a certain peripheral. *p3_xxx* is used throughout this document as generic peripheral name.

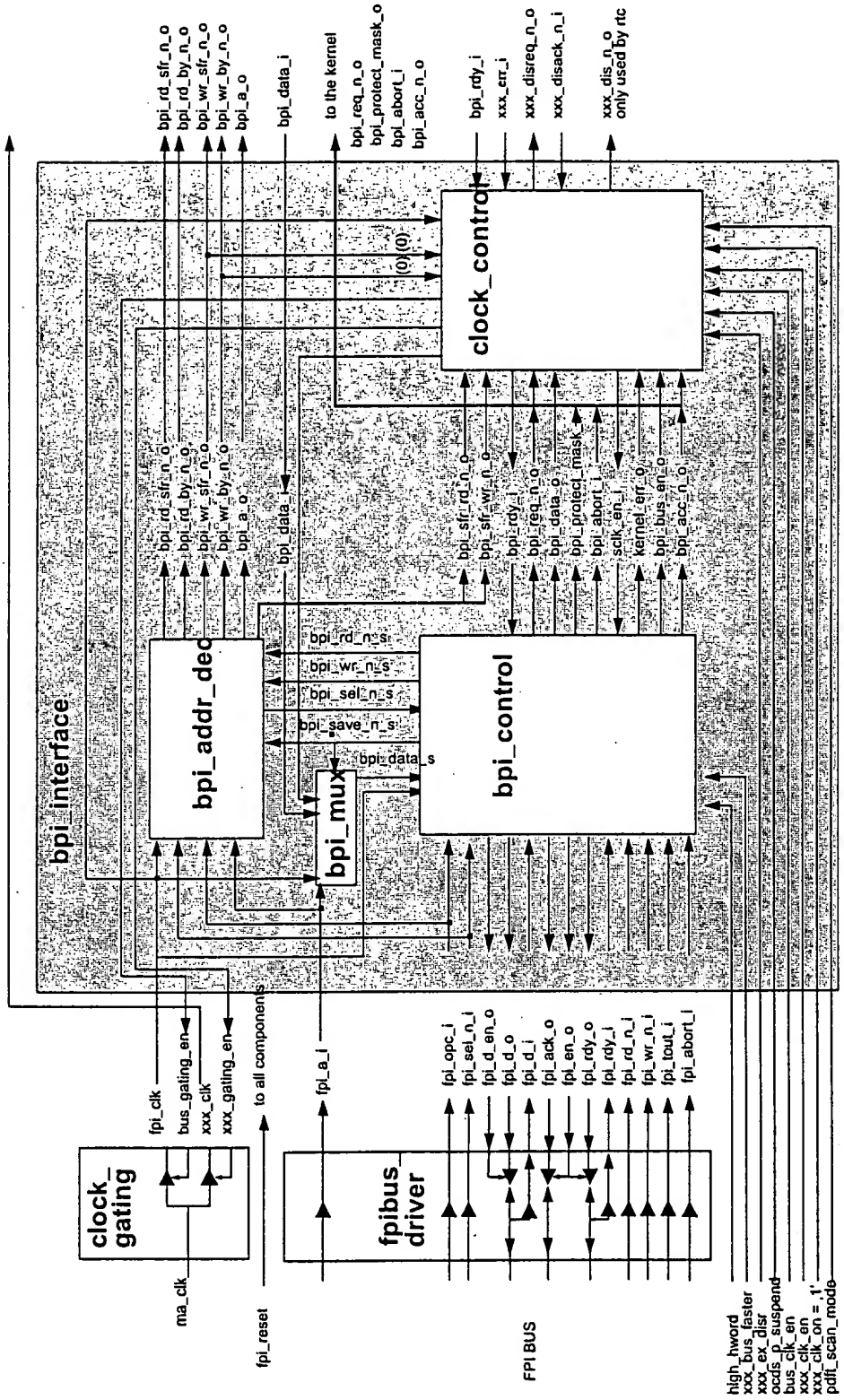


Figure 1: BPI architecture

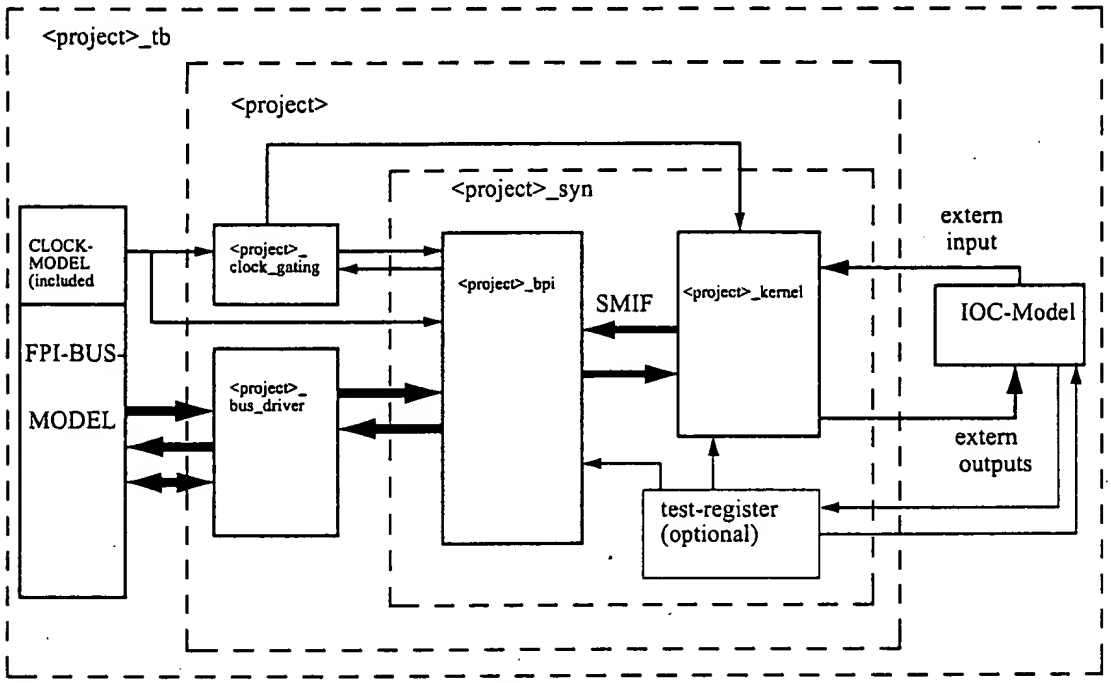


Figure 2: Architecture of the whole peripheral and testbench

If you use this example proceed as follows:

- Complete the component <project> and<project>_syn with the external input and output signals given by your<project>_kernel.
- Extend the signal list and modify the port map in file <project>_tbe_tba-a.vhd.
- Change the constant nr_of_pins_c and edit the constant pin map of the <project>_IOC_top port map

The <project>_tbe_tba-a.vhd contains an example.

3. Parameter setting in the project package

The configuration of the BPI is handled in the BPI_PACKAGE, contained in the file <project>_bpi_pack-p.vhd.
For every peripheral this package needs to be copied into the project packages directory under a unique name, usually by prefixing the original name with the project prefix, e.g. *P3_XXX*_BPI_PACK-P.VHD. Configuration of the BPI is then accomplished by editing (extending) this file. One important point is editing the name of the package likewise !

> Example for the watchdogtimer (WDT):
> $HWPROJECT/VHDL_PACKS/RTL/P3_WDT_BPI_PACK-P.VHD.

The personalised project package is then imported by all other units with the statement

USE WORK.*P3_XXX*_BPI_PACK.ALL;

Inside this package several constants are to be checked and changed if necessary:

fpi_addressbuswidth_c	Indicates the width of the incoming FPI_A - BUS Preset to 32 Bit. Normally no change necessary
fpi_databuswidth_c	Indicates the width of the bidirectional FPI_D - BUS Preset to 32 Bit. Normally no change necessary
kernel_databuswidth_c	Indicates how many bits are provided by the largest peripheral register in one access. Possible values are 32, 16, 8. The constant sfr_size_c indicates the numbers of bytes and is calculated from kernel_databuswidth_c by: sfr_size_c := kernel_databuswidth_c / 8
addressbuswidth_c	The constant addressbuswidth_c indicates how many bits are to be decoded by the address decoder for the SFR select signals. In order to get the correct address of the register the bits fpi_a(fpi_high_c downto fpi_low_c) will be decoded. Note that fpi_a(1 downto 0) are reserved for the byte selection. The default value is 8; {(7 downto 0)}
fpi_low_c	LSB of the decoded address for the SFR addressrange. Preset to 2. Normally no change necessary
fpi_high_c	MSB of the decoded address for the SFR addressrange. Preset to 7. Normally no change necessary
handshake_c	When you need a handshake-procedure between interface and kernel, set the constant handshake_c:=1; Preset to 0
fpi_addr_width_c	Range for bit_vector. Preset to 8. No change necessary unless the constant addressbuswidth_c was changed. This constant is only used to define the type of the following constant sfr_addr_c.
sfr_addr_c	Array of SFR-Addresses for the address decoder. Note that the first 4 addresses (x'00,x'04,x'08,x'0C) are predefined !
destructive_read_c	Indicates whether the peripheral includes destructive read registers. Preset to 0

dest_addr_c	Subarray of SFR-Addresses (must also be present in sfr_addr_c) which are destructive read registers. Preset to „10“. This constant is only relevant if destructive_read_c = 1. **If only one register is dest.read insert it twice**

The next group of constants is relevant if a RAM-interface is required only.

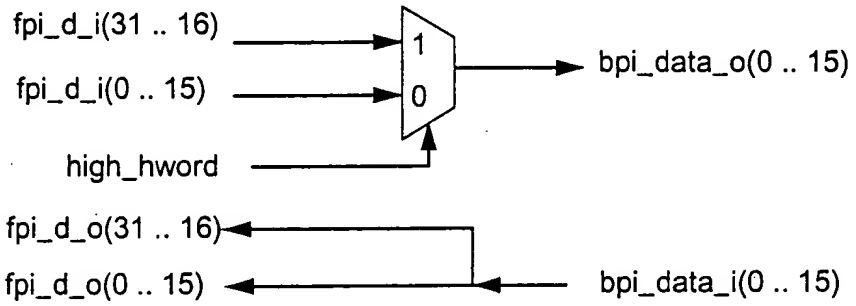
ram_interface_c	Gobal indication if a RAM is required! Preset to FALSE
ram_addr_c	The constant indicates the bit combination which shall create the select signal for the ram inside the kernel. Preset to „1“ This constant is compared to fpi_a(ram_high_c downto ram_low_c) **If ram_high_c is not equal to ram_low_c define a subtype !**
ram_addr_low_c	Indicates the lsb of the RAM address. Preset to 0
ram_addr_high_c	Indicates the msb of the RAM address. Preset to 7 bpi_a<=fpi_a(ram_addr_high_c downto ram_addr_low_c) Preset to bpi_a<=fpi_a(7 downto 0)
ram_low_c	LSB of the decoded ram address. Preset to 8. Normally no change necessary
ram_high_c	Preset to 8 ; -- only 256 byte RAM If you set this constant, for example to the value 9 then don't forget to change the constant ram_addr_c to a 2-Bit-Vector. Example: std_ulogic_vector(ram_high_c downto ram_low_c) := „10“ This sector indicates if a RAM or SFR are selected.
ram_databuswidth_c	Preset to 32 -- the same as the FPI data bus. You can change it to 16 or 8 for another ram databuswidth. **!! IN THE CURRENT VERSION !!!!** **!! ram_databuswidth_c must equal kernel_databuswidth_c!** **!! It is no own data input for the RAM inserted !!**
ram_access_c	The result of the calculation ram_databuswidth_c / 8

If a RAM-module is not configured (ram_interface_c = FALSE), the RAM special signals must not be used in the kernel. These are

BPI_A_I	the registered fpi_a(ram_low_c-1 downto 0)
BPI_WR_N_I	write signal for the RAM
BPI_RD_N_I	read signal for the RAM
BPI_CS_N_I	chipselect signal for the RAM

4. Binary and Enhanced Mode

For compatibility reasons some bus mapping is provided. This is controlled by the special signal high_hword. Only if the FPI databus width is 32 and the Kernel databuswidth is 16 is it possible to multiplex the upper or the lower16 Bit of the FPI databus to the kernel databus independently from the lsbs of fpi_a ! Nomally this signal must be held '0'.



5. The FSM of the controlunit

5.1. control_m1

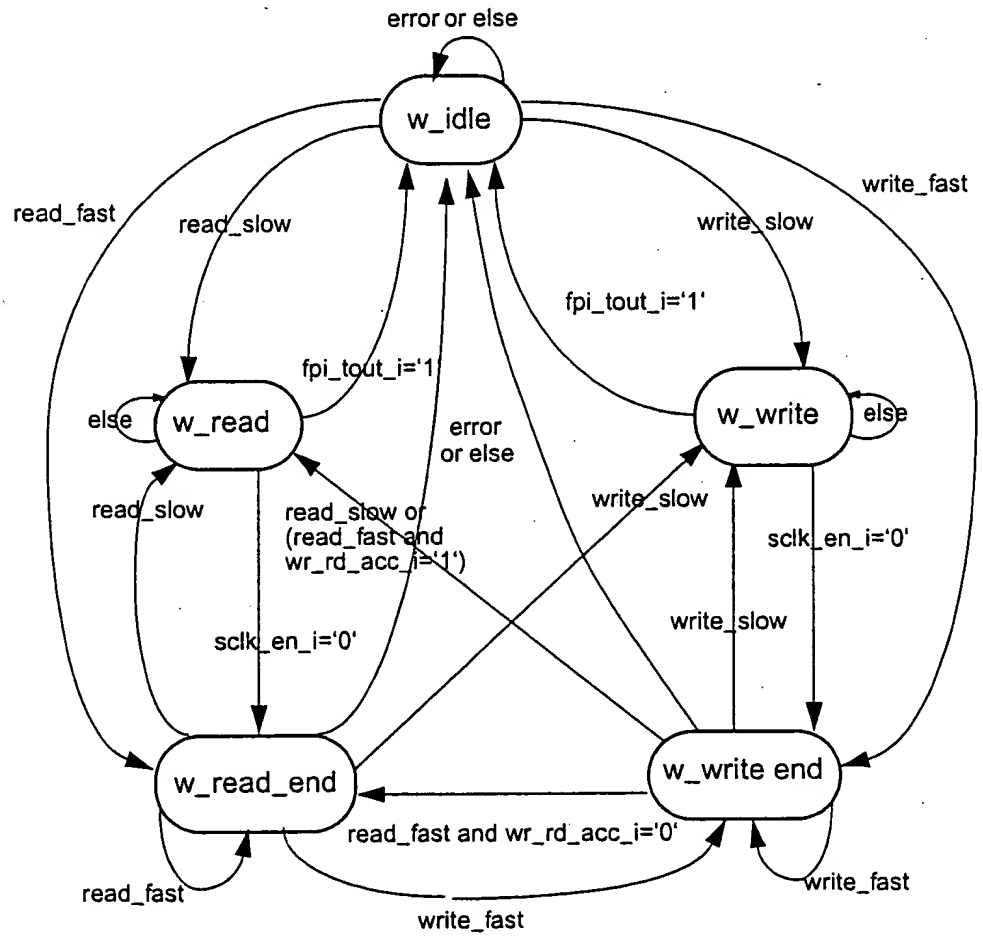
read_fast <= xxx_bus_faster_i='0' and fpi_rd_n_i='0' and fpi_opc_i /= „1111" and
fpi_sel_n_i='0' and fpi_rdy_i='1' bpi_sel_n_i='0';

write_fast <= xxx_bus_faster_i='0' and fpi_rd_n_i='1' and fpi_opc_i /= „1111" and
fpi_sel_n_i='0' and fpi_rdy_i='1' bpi_sel_n_i='0' and fpi_wr_n_i='0';

read_slow <= (xxx_bus_faster_i='0' and fpi_rd_n_i='0' and fpi_opc_i /= „1111" and
fpi_sel_n_i='0' and fpi_rdy_i='1' bpi_sel_n_i='0')
or (read_fast and bpi_dest_i='1';

write_slow <= xxx_bus_faster_i='0' and fpi_rd_n_i='1' and fpi_opc_i /= „1111" and
fpi_sel_n_i='0' and fpi_rdy_i='1' bpi_sel_n_i='0' and fpi_wr_n_i='0';

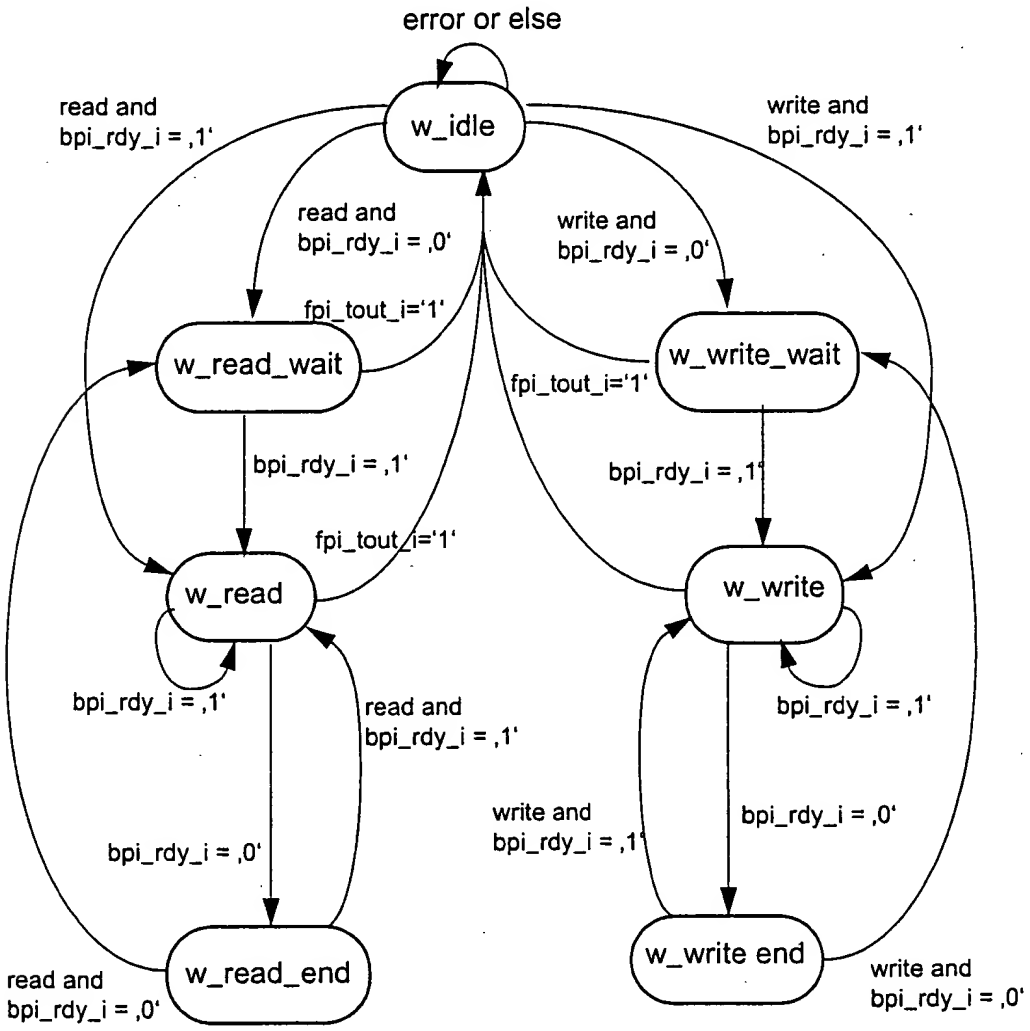
error <= fpi_tout_i='1' or kernel_err_i='1';



5.2. control_h (handshake)

read <= fpi_rd_n_i='0' and fpi_opc_i /= „1111" and
fpi_sel_n_i='0' and fpi_rdy_i='1' bpi_sel_n_i='0'

write <= fpi_rd_n_i='1' and fpi_wr_n_i='0' and fpi_opc_i /= „1111" and
fpi_sel_n_i='0' and fpi_rdy_i='1' bpi_sel_n_i='0'



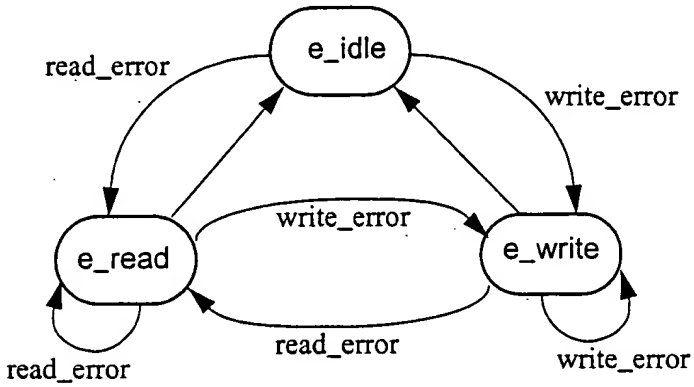
5.3. FSM for error handling

An error occured if

- the peripheral is selected but the SFR address is wrong,
- a access is performed while the kernel clock is switched off or
- a access is performed while the kernel indicates a kernel error

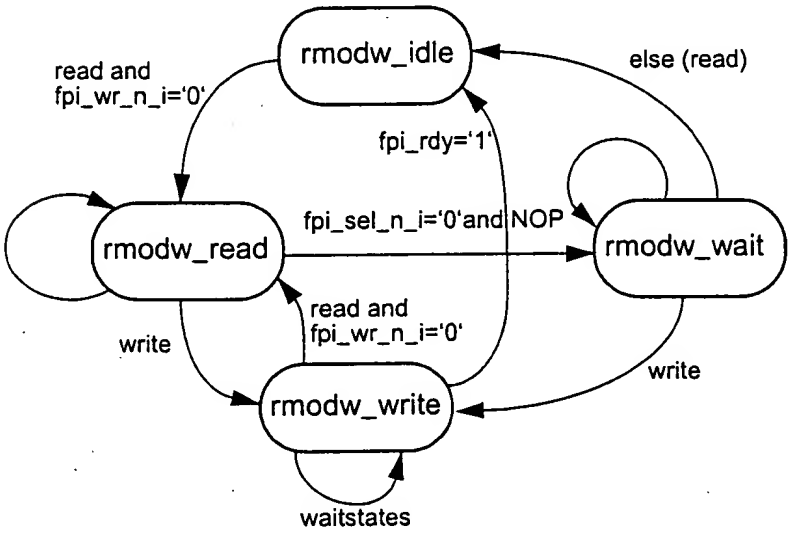
In the e_read state the fpi data bus nonetheless must driven by the peripheral.
If the kernel clock is switched off only accesses to the clock control register is allowed.

In case of timing problems there is a constant error_handling_c **defined inside the architecture p3_xxx_bpi-rtl-a.vhd to switch off this errorhandling** (constant error_handling_c: boolean:=false;)! **In this case the fpi_ack is in everytime NSC!**

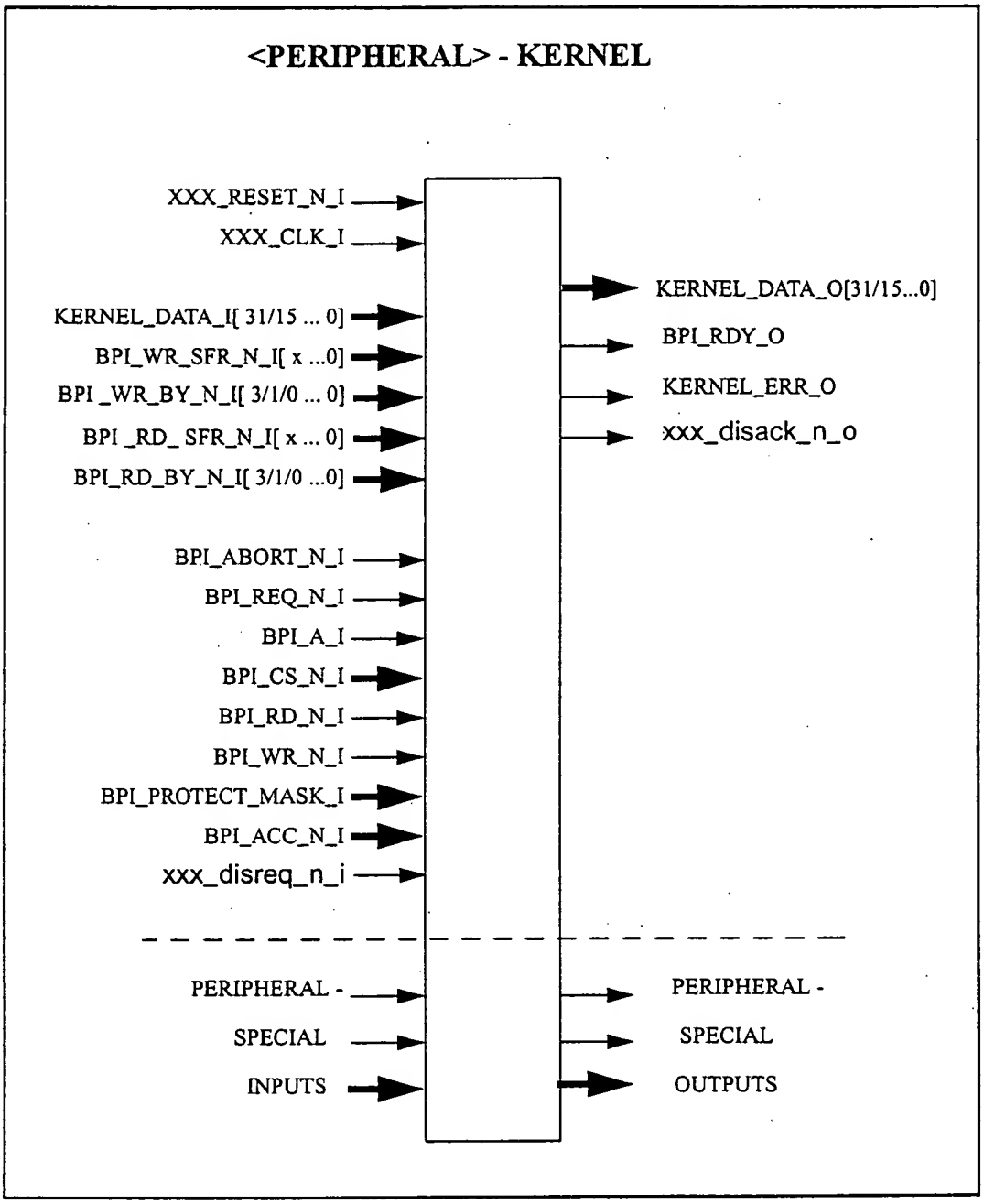


5.4. The read modify write FSM

The rmodw FMS rememberes the read part of a modify access and waites for the corresponding write access.



6. I/O signals from Peripheral Kernel



The signals have already been described above.
If no error indication from the kernel is required, two signals must be driven nonetheless by the architecture of the kernel:

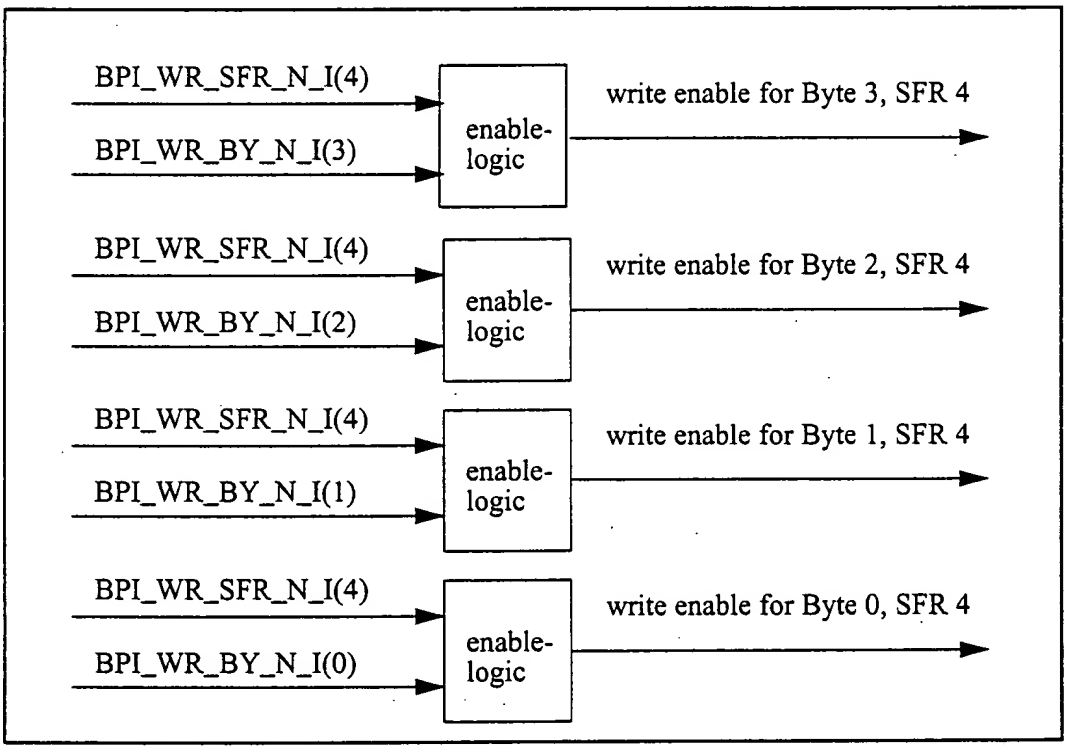
```
kernel_err_o <= ‚0'; -- special error from kernel
xxx_disack_n_o <= xxx_disreq_n_i; -- always ok to disable clock
```

7. How you can use the port signals

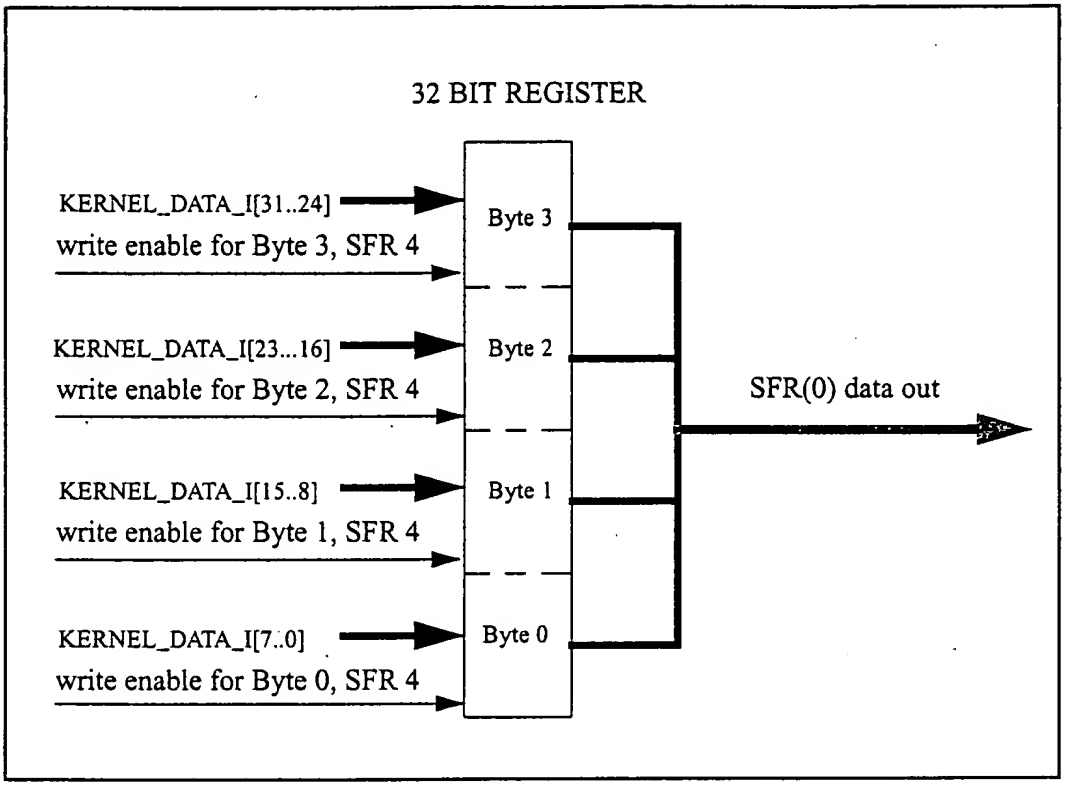
7.1. Write enable logic

A register normally consist of one, two or four bytes. In order to write each byte separately the BPI provides a dedicated enable signal (BPI_WR_SFR_N_I(x)) for each SFR and a separate enable signal (BPI_WR_BY_N_I(3..0)) for each byte of the data bus. For each byte of each register the write enable will be generated from this signals. If the signal BPI_WR_SFR_N_I of the SFR and all signals of the BPI_WR_BY_N_I are low, then the whole register is enabled to store the new data.

The figure below illustrates the proposed circuit

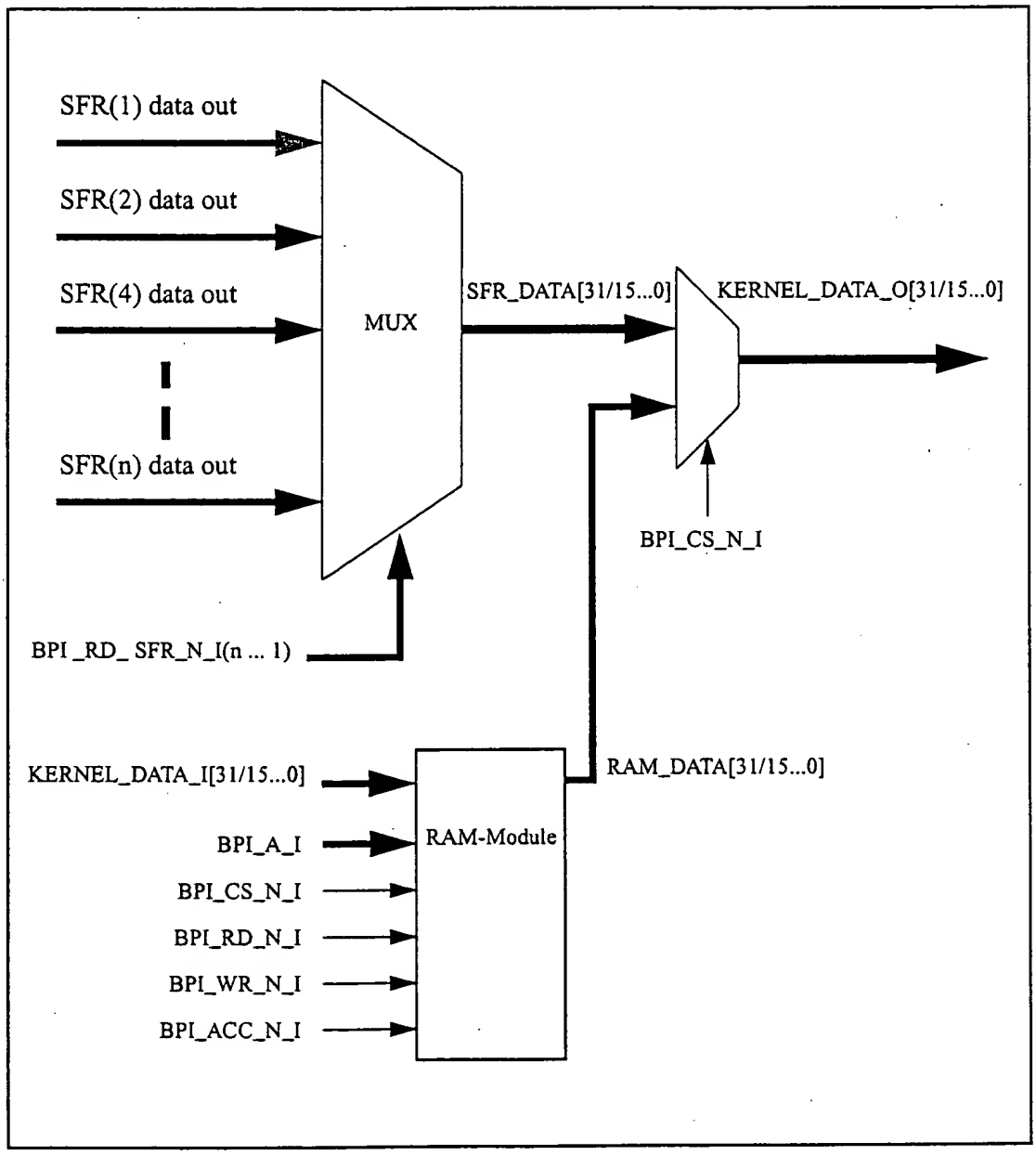


Caution!!
The BPI_WR_SFR_N_I[0 .. 3] and BPI_RD_SFR_N_I[0 .. 3] are reserved normally !!
SFR(0) is the clock control register and is instantiated in the BPI.
SFR(1) is the peripheral input select register
SFR(2) is the identification number
SFR(3) is reserved for future use
The correlation between index and address is given by the order of the addresses in the addresslist in the bpi_package.
Note that only SFR(0) is implemented inside the BPI's architectures (clc-entity)


32 BIT REGISTER
KERNEL_DATA_I[31..24]
write enable for Byte 3, SFR 4
KERNEL_DATA_I[23...16]
write enable for Byte 2, SFR 4
KERNEL_DATA_I[15..8]
write enable for Byte 1, SFR 4
KERNEL_DATA_I[7..0]
write enable for Byte 0, SFR 4
Byte 3
Byte 2
Byte 1
Byte 0
SFR(0) data out

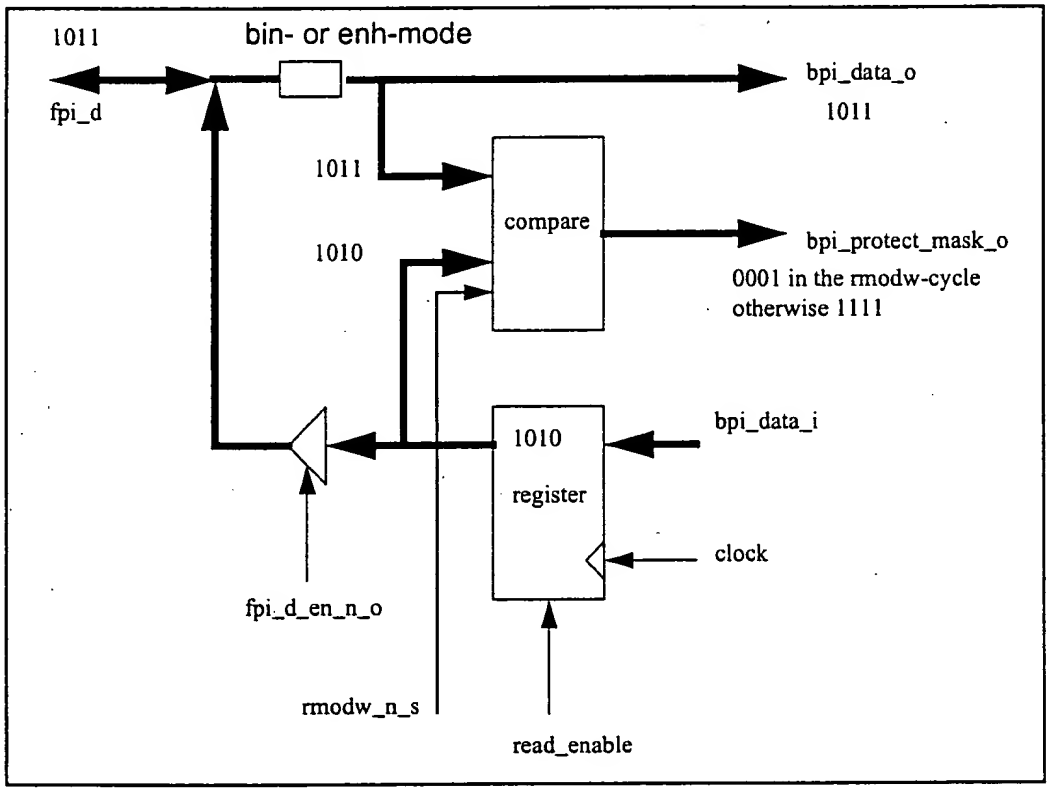
7.2. Multiplex output data

The next figure shows one possibility to propagate the output data of the kernel. Note that multiplexing RAM data and SFR data in two stages is only an example.

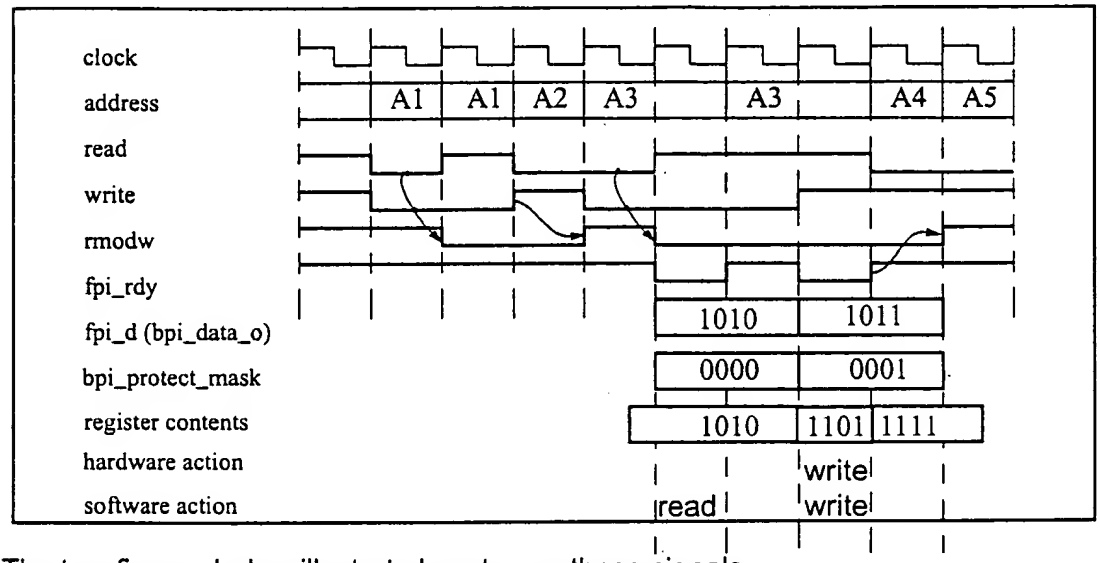


7.3. Read modify write cycle and protect mask

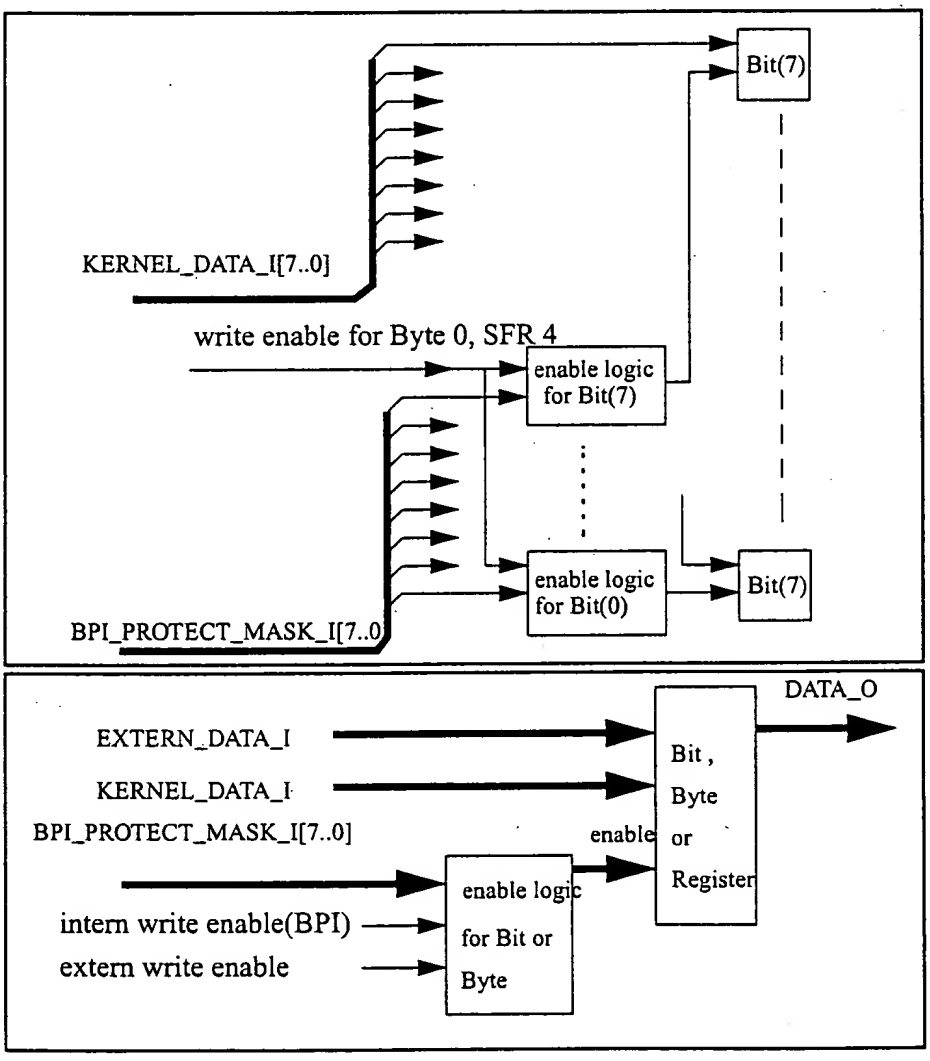
Read-Modify-Write is coded on the FPI bus and recognized by the BPI interface. The read modify write signal is not forwarded to the kernel.
As a result a signal RMODW_N_S is low active for the complete read modify write cycle on the FPI-BUS. As no other SFR may be read intermittingly the read data can be stored in the read register. A protection mask can now be derived by comparing the (locally stored) read data with the new data to be written. All bits not changed are given a mask value 0 if the RMODW_N_S signal is active.
In a normal write or read cycle, all bits of the mask are „1".
Protect mask bit equal to „1" means that the data bit must be written into the register.
Protect mask bit equal to „0" means that the data bit must not be written into the register.
This is illustrated by the next figures.



The bpi_data_i are only captured at the end of the read-cycle and remain unchanged by the write- and idle-cycles on the FPI bus. The protect mask is provided every time. Only register bits which can be modified by SW and HW need to use the protection mask.



The two figures below illustrate how to use these signals

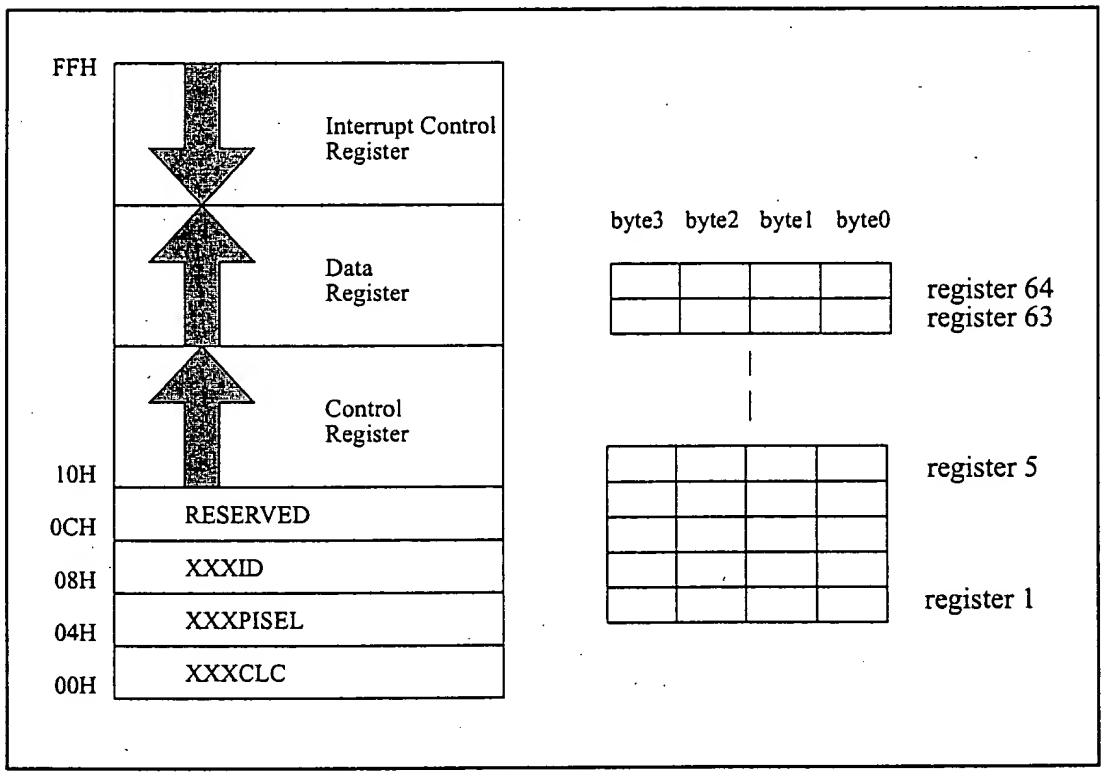


8. Peripheral Address Map

Each peripheral has n x 256 Byte register blocks or n x 64 32-Bit register. The register is selected with signals BPI_WR_SFR_N_I or BPI_RD_SFR_N_I . Four registers are fixed:

- Address 00H is the peripheral clock control register.
- Address 04H is the peripheral port input select register.
- Address 08H is the peripheral identification. No real register, hard coded.
- Address 0CH is reserved.

Control registers start at address 10H, followed by the data register.
The interrupt control register start at the address FFH decreasing order.



9. Usage of the fixed Registers

9.1. XXXID - REGISTER

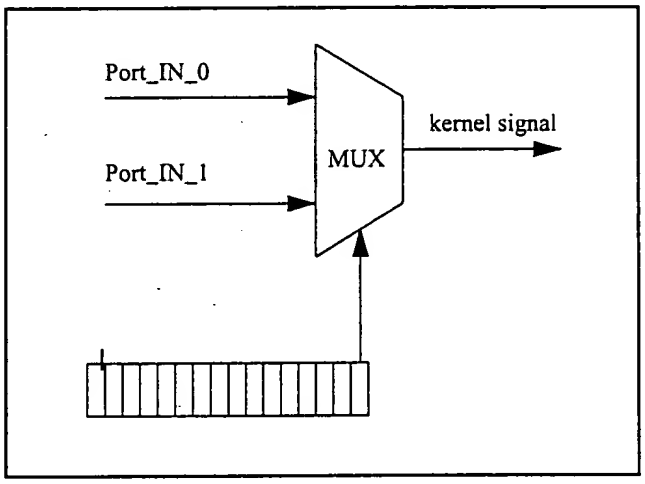
The XXXID is not a real register, it is hard coded and readable only.
Example :

```
perid_s <= „10100111“;
        or
perid_c : std_ulogic_vector(8 downto 0) := „10100111“;
```

It is selected with bpi_rd_sfr_n_i(2).
It should be assured by appropriate dont_touch attributes that the ID is implemented on silicon as matel contacts. That allows cange of revision for metal redesigns.

9.2. XXXPISEL - REGISTER

The XXXPISEL register indicates which port shall be used for kernel inputs.
The register has to be implemented by the peripheral designer and is selected with bpi_wr_sfr_n_i(1). Every bit in this register controls one multiplexer.
Therefore, every external input of the kernel can be driven by one of two sources.
Example :
If XXXPISEL(0) = ‚0' then kernel signal <= Port_0 else kernel signal <= Port_1;



9.3. XXXCLC - REGISTER

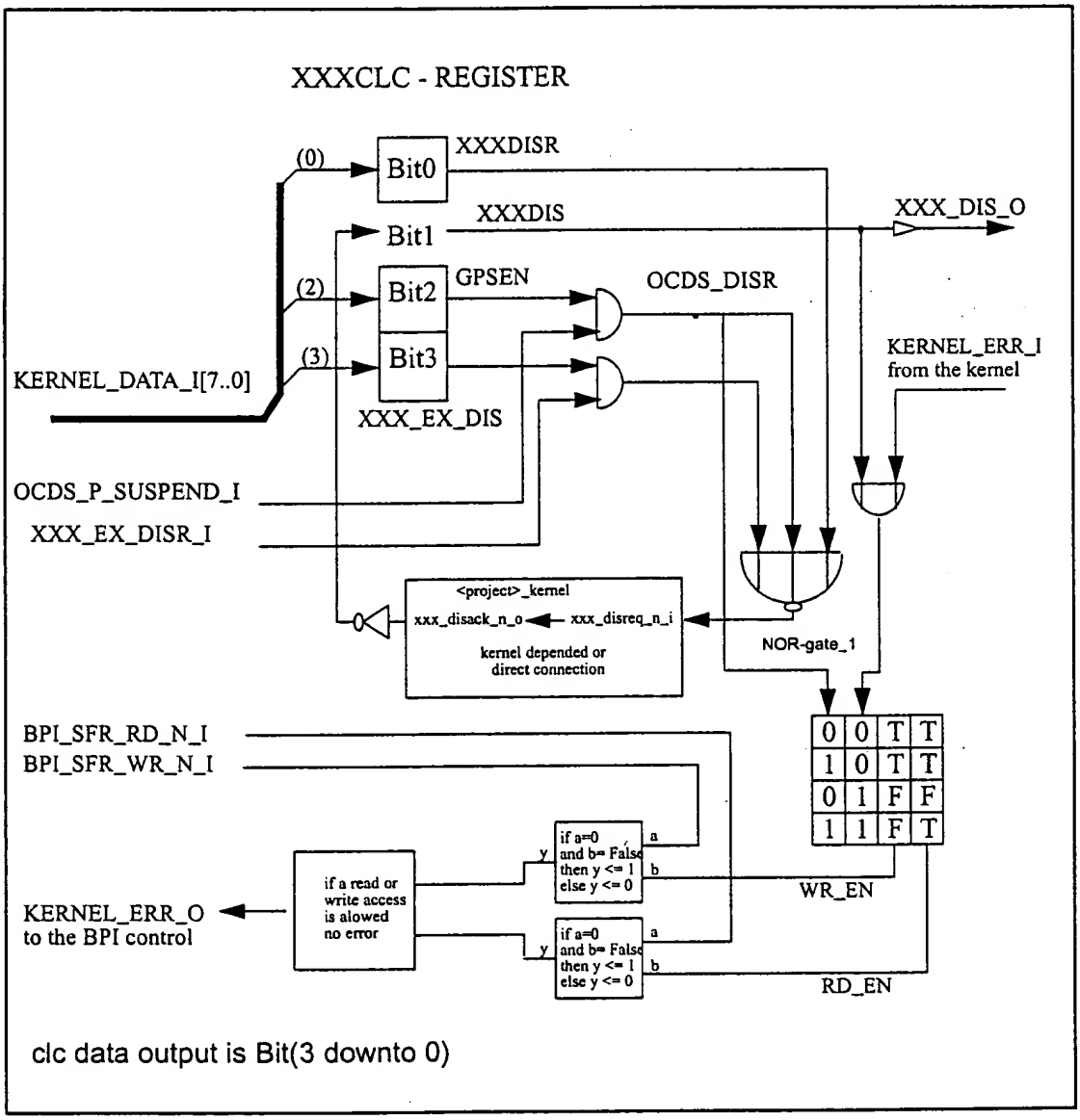
The state of the peripheral clock is controlled by a register bit "XXXDISR" located in the register XXXCLC of the peripheral core. This register XXXCLC is clocked with the bus clock to be able to switch the peripheral clock on again if it was off. If required by the peripheral's functionality switching off the clock can be prevented by the peripheral. The actual clock state will be shown by the state bit "XXXDIS" within the same register. The signal "XXX_DIS_N" which is a combinatorial combination of other clock control signals represents the actual enable state of the peripheral clock and is used to switch on/off the peripheral clock. The clock gating buffer is located in the Clock Gating block of a peripheral. The clock enable signals used to control the clock speeds are generated in the central Clock Generation and distributed to the Clock Gating block.
The BPI module rejects every FPI bus access with Error condition if the peripheral clock is switched off (signaled by the signal Kernel_err_o- driven by clc module, not peripheral kernel).

9.3.1. Coherence between XXXCLC and the OCDS_P_SUSPEND, XXX_EX_DISR

For On Chip Debugging support an additional signal OCDS_P_SUSPEND is intro-

duced to stop the peripheral clock for debugging if this function is enabled. If debugging mode is active the peripheral core rejects write access to registers connected to the peripheral clock by activating the signal Kernel_err. This causes the BPI module to reject FPI bus accesses with Error condition. Read accesses to registers of the peripheral clock domain are possible.
To be compatible with old products an XXX_EX_DISR signal is introduced to disable the peripheral clock.

For more information see also the **Peripheral Clock Strategy Specification**



10. Forwarding

If a read access immediately follows a write cycle, it is normally not possible to read the previously written data in a zero waitstate access due to the required synchronisation. To resolve this problem no forwarding multiplexer is provided; instead a waitstate will be inserted if a read access is performed after a write access to the same register and without any idlestates in between !

11. Timing Diagrams

As there is no fixed relationship between bus clock and peripheral clock several different timing diagrams are provided.

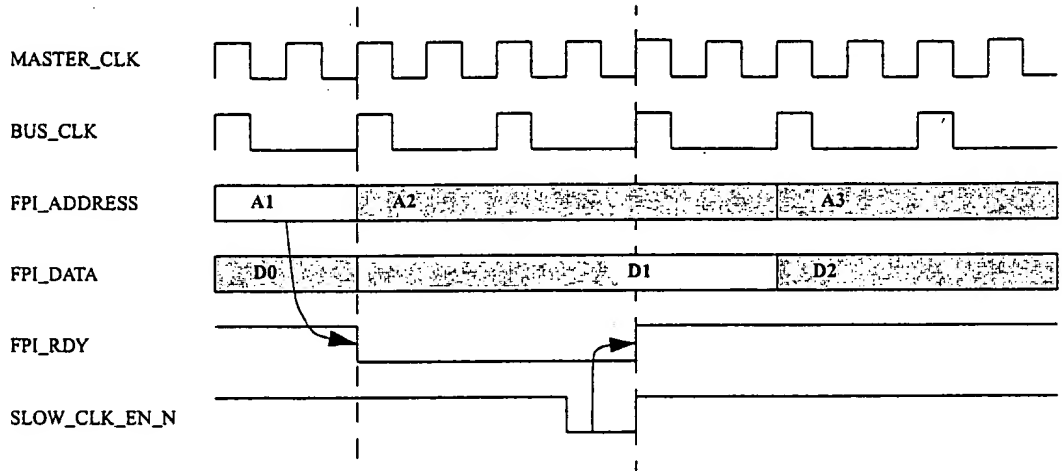
The first possibility is, that the bus clock is faster (bus_faster is ‚1') as the peripheral clock.

The second possibility is, that the bus clock is slower tahn or exactly as fast as the peripheral clock (bus_faster is ‚0').

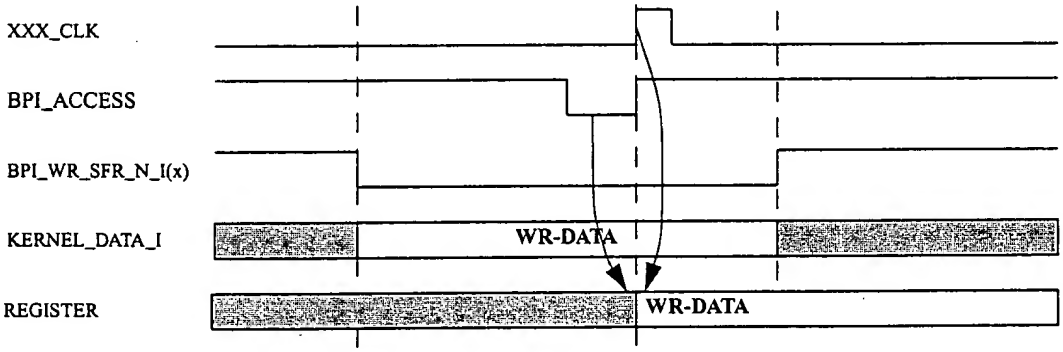
If it's necessary to implement a RAM inside the kernel the cycle must start with one waitstate. This is not implemented!

The BPI_ACC_N_I signal is used to avoid multiply writes to the same address due to to fast a peripheral clock. It will be driven low activ one clockcycle before the relevant rising edge of the peripheral clock. It is activated whenever a register must store the data in the write cycle or change the data after a destructive read cycle.

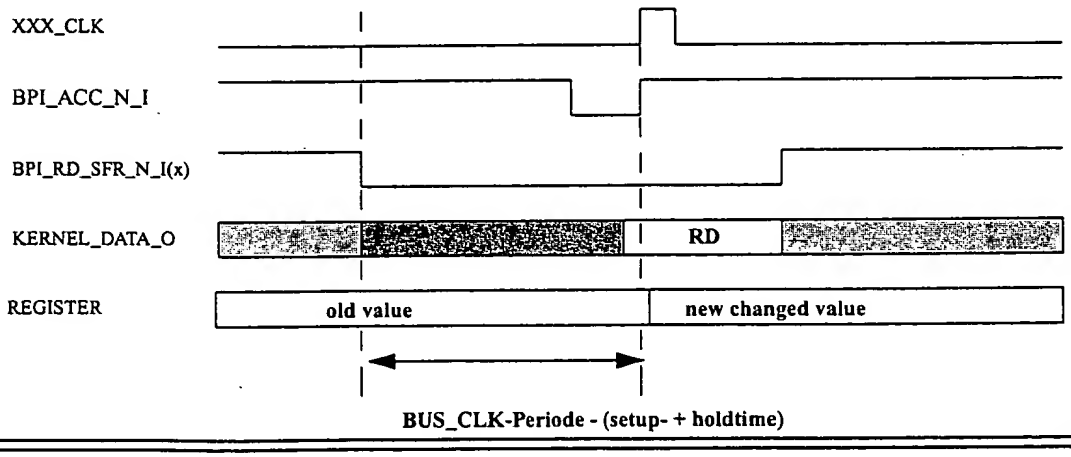
BUS_CLK FASTER THAN XXX_CLK



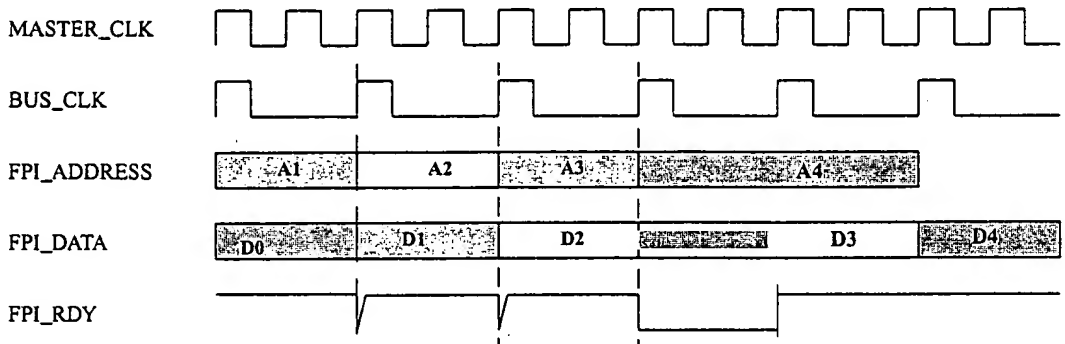
NORMAL WRITE , >= ONE WAITSTATE



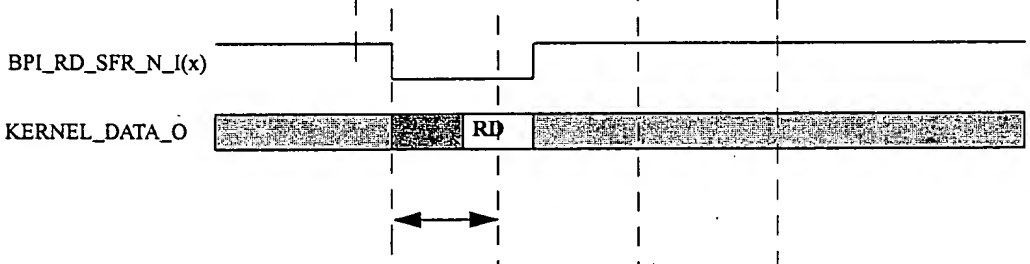
DESTRUCTIVE READ , >= ONE WAITSTATE



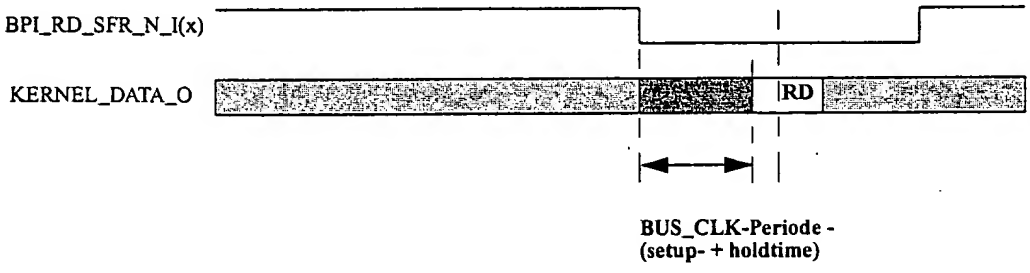
BUS_CLK FASTER THAN XXX_CLK



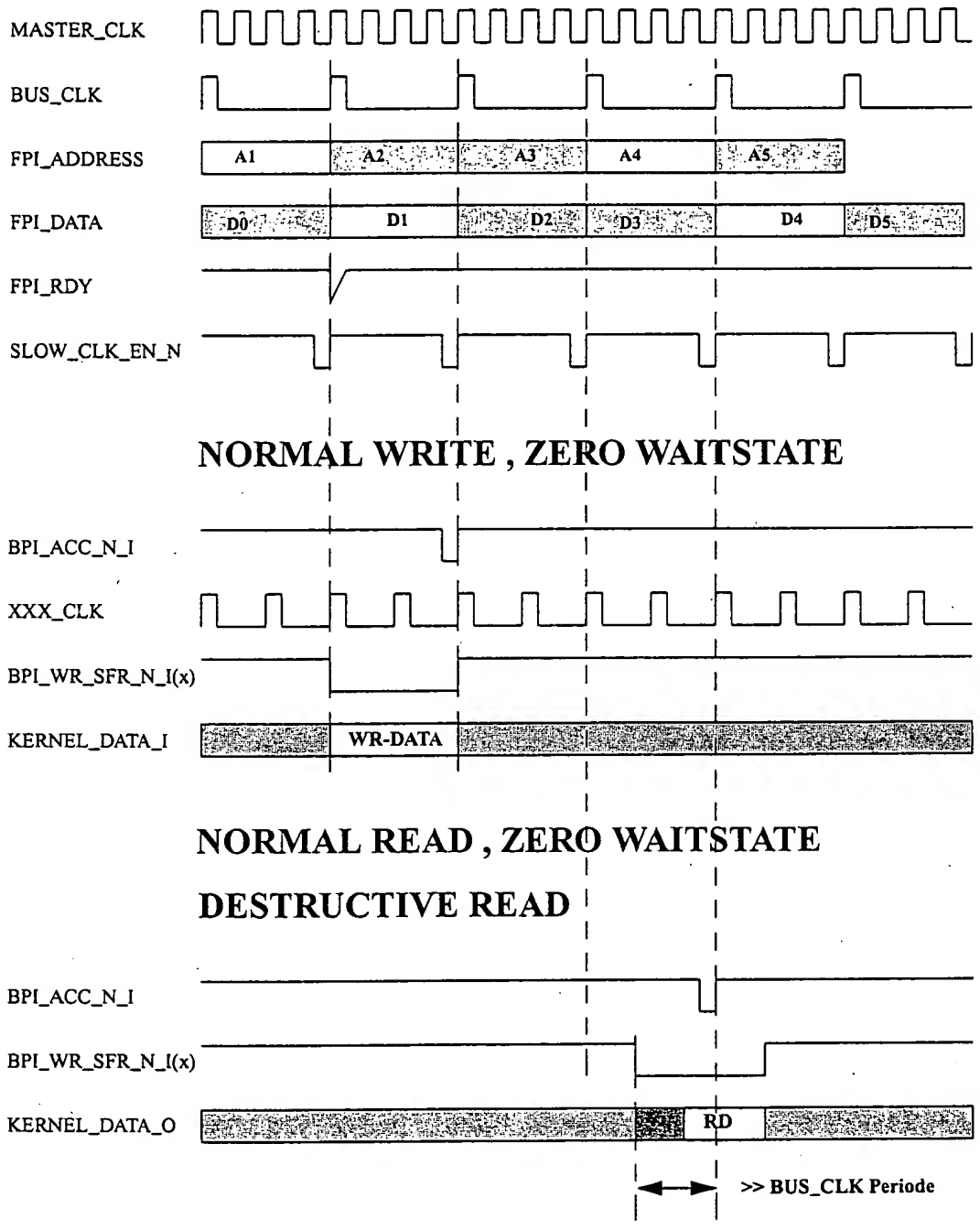
NORMAL READ , ZERO WAITSTATE



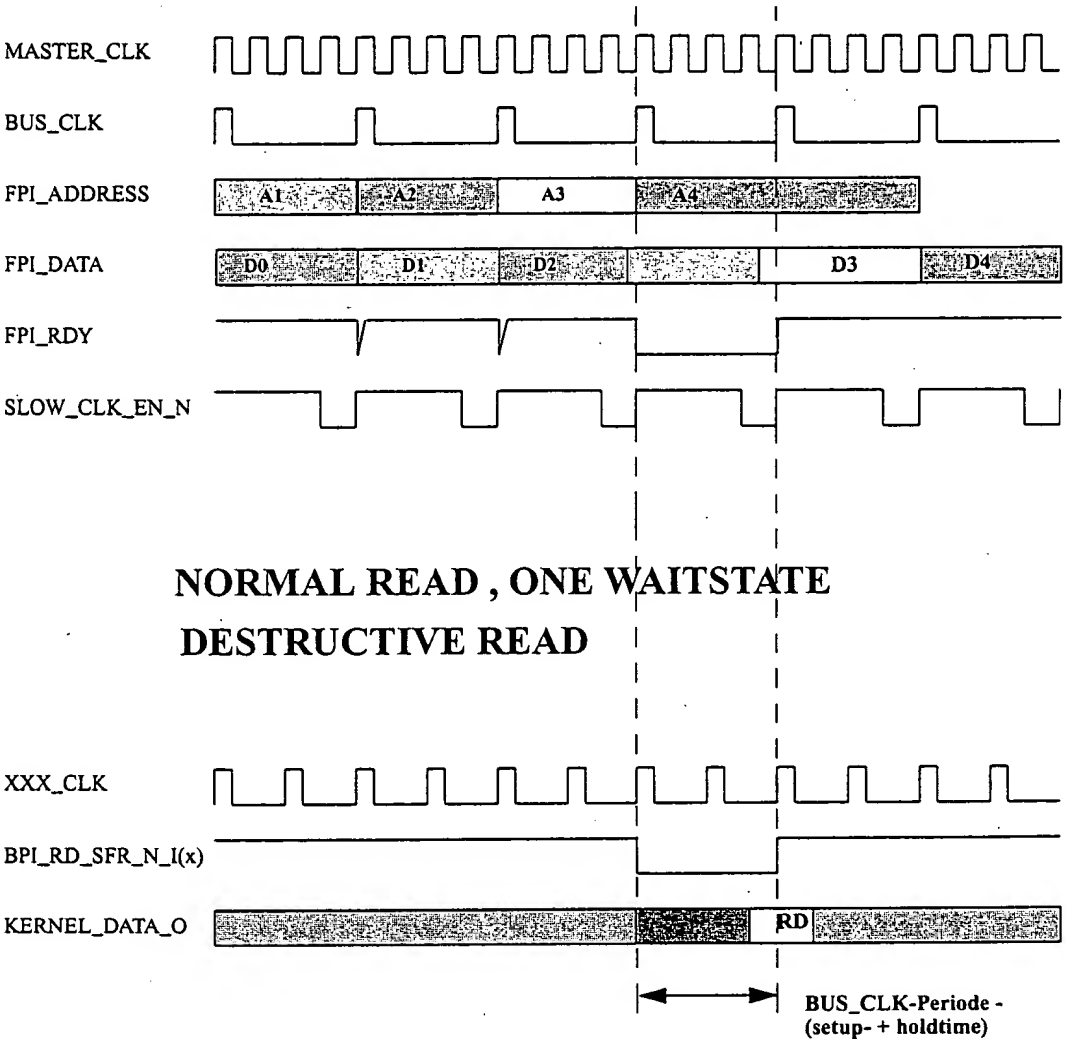
NORMAL READ , ONE WAITSTATE



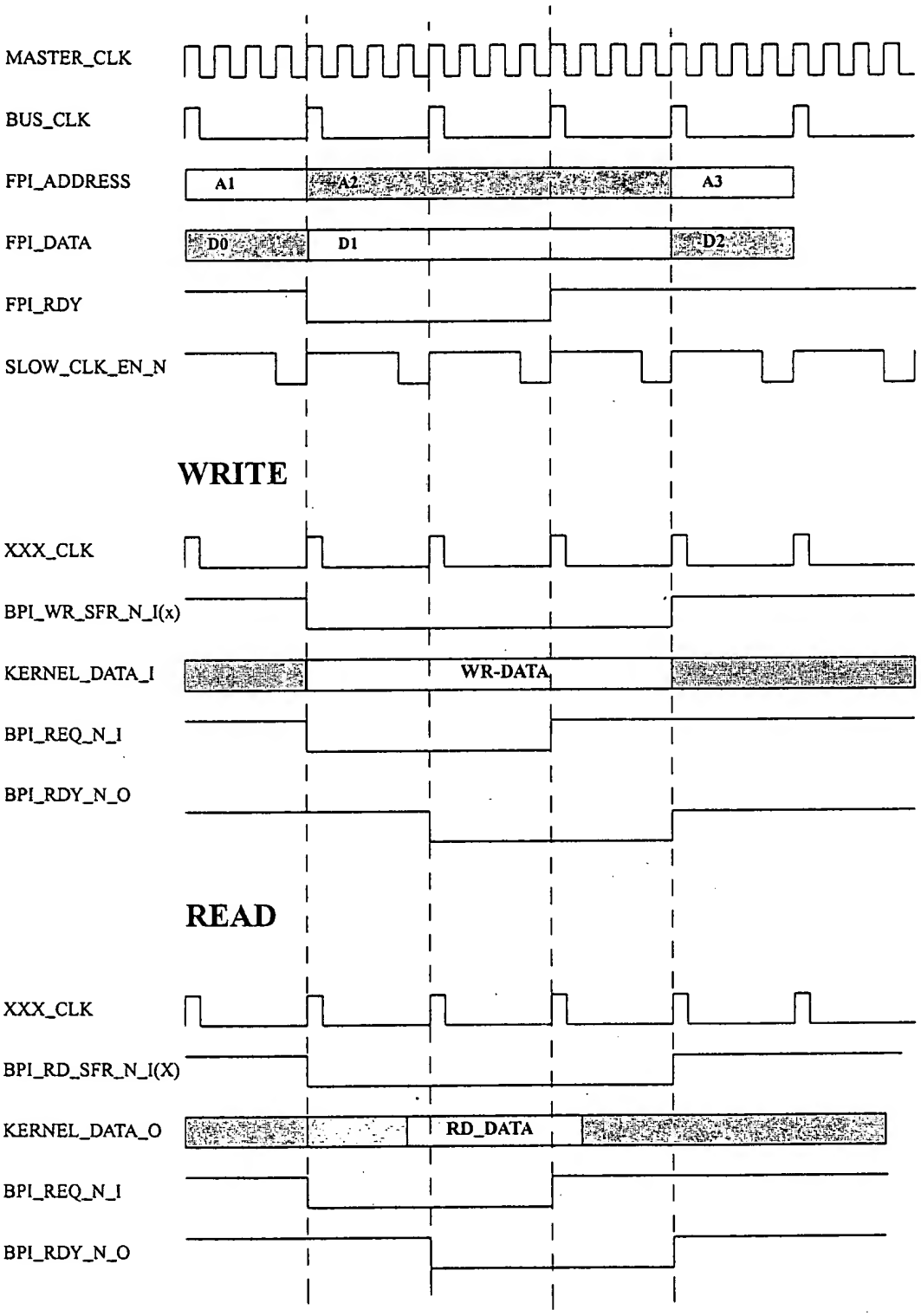
BUS CLK SLOWER OR EQUAL THAN XXX CLK



BUS_CLK SLOWER OR EQUAL THAN XXX_CLK

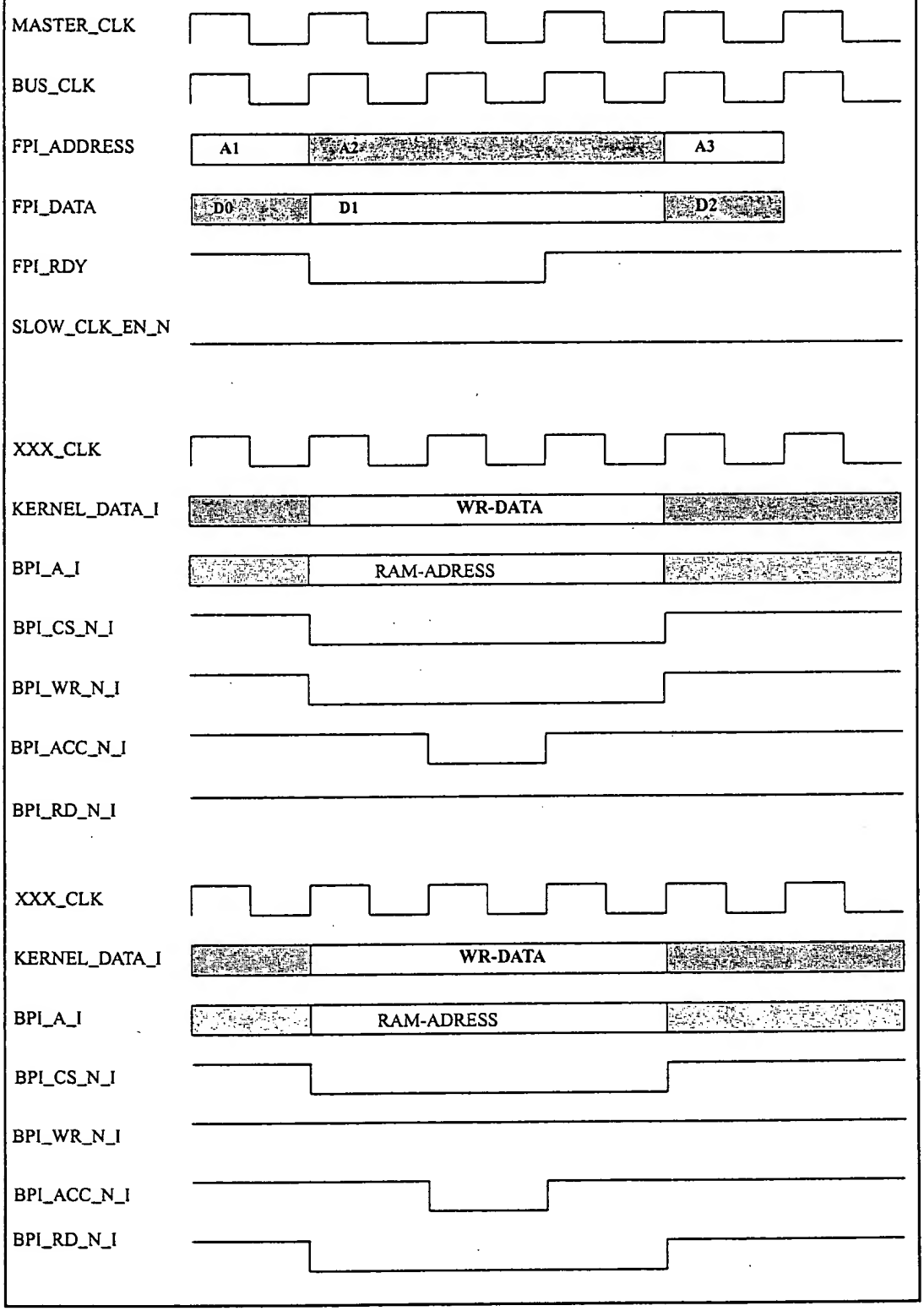


HANDSHAKE BETWEEN INTERFACE AND KERNEL

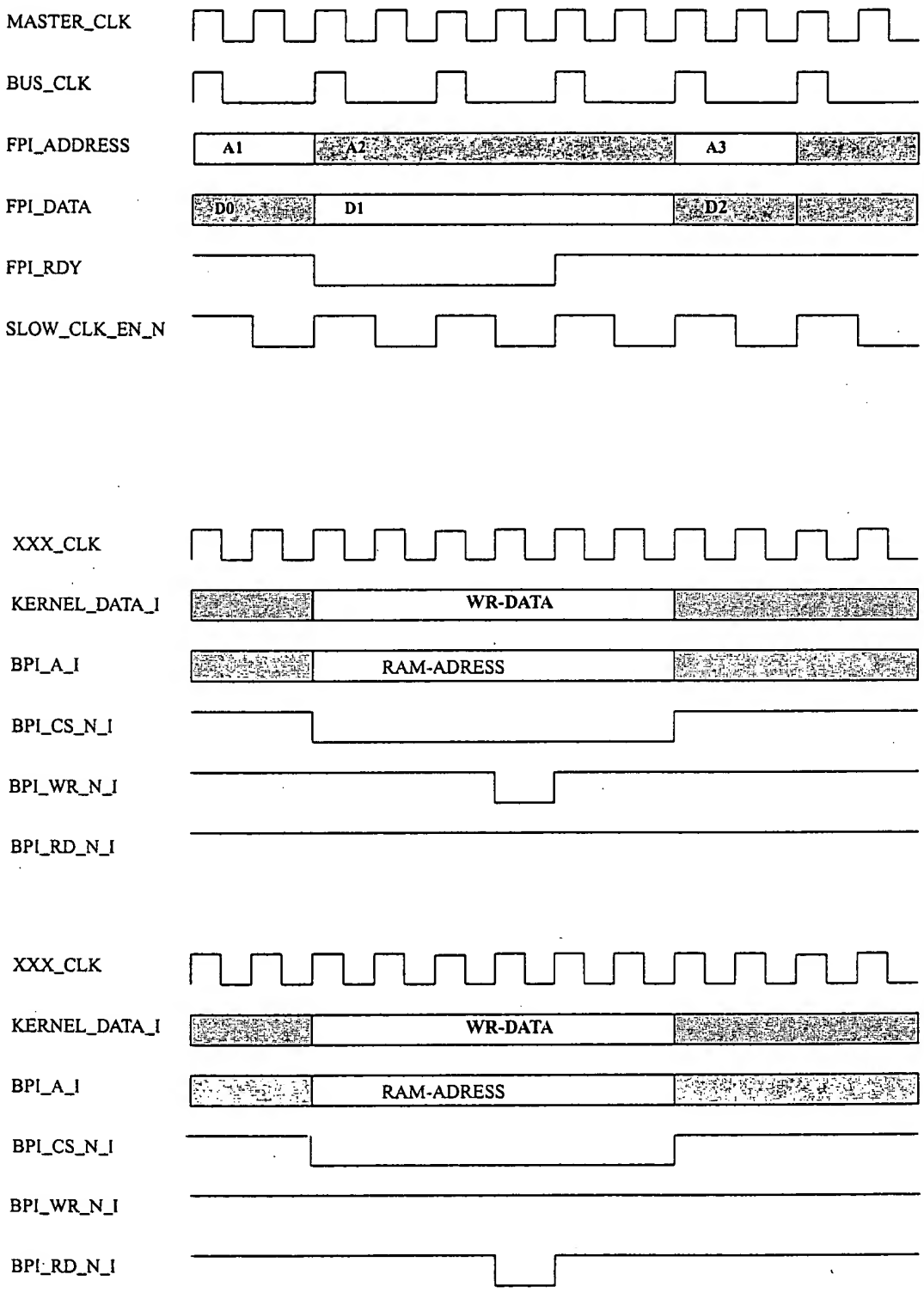


RAM INTERFACE (not implemented)

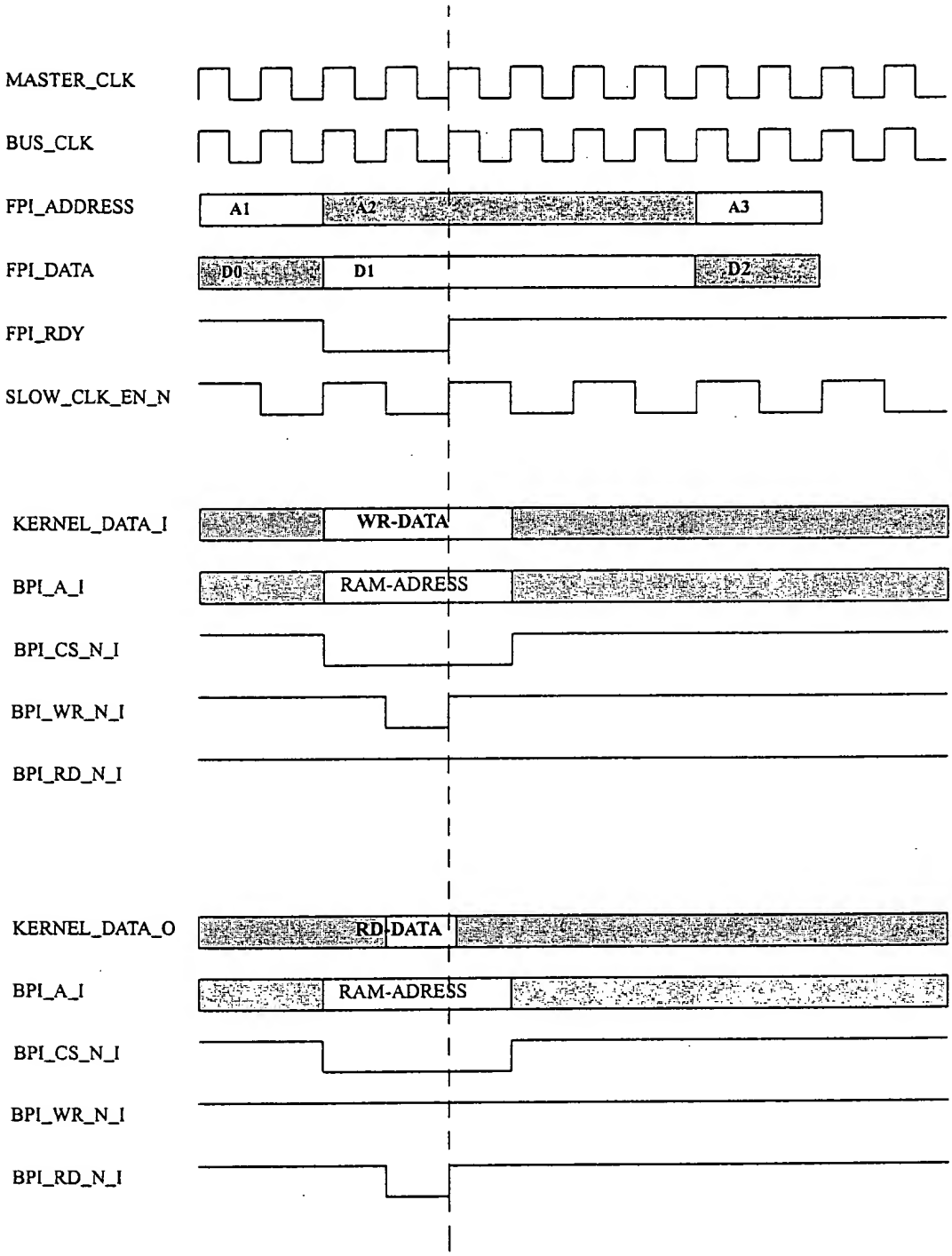
The current version is like a nomal access and provides the RAM signal but not in the timing below



RAM INTERFACE - BUS SLOWER THAN KERNEL



RAM INTERFACE - BUS SLOWER THAN KERNEL



12. Module Developers Test Cook Book

This chapter is a copy of Hermann Obermeir's paper regarding this topic; contact him for updates !

Platform peripherals are prepared for ATPG and for isolated module test. The following gives a brief overview what a module developer has to do.

12.1. Automatic Test Pattern Generation(ATPG)

12.1.1. Prepare setup files

e.g. GNUmakefile.setup
INSERT_SCAN_DESIGN := true
SCAN_CHAINS := <desired_number>

12.1.2. Run check test

- In Design Analyzer: Tools -> Test Synthesis: Check Design Analyzer
- you should have no violations

12.1.3. Insert Scan Registers and generate pattern

Use "ssemake insert_scan" to insert scan registers.
Build scan chains up to a maximum length of 100 flipflops.
serial inputs of the scanchains: **pdft_sci_0_i, pdft_sci_1_i,.....**
serial outputs: **pdft_sco_0_o, pdft_sco_1_o,....**
scan enable input: **pdft_scen_i**

12.1.4. Generate test Pattern

"ssemake atpg" You should achieve 100% fault coverage.
(If not make sure with a fault simulation of your module test pattern, that the untested faults are tested with your module test pattern)

12.1.5. Save Results

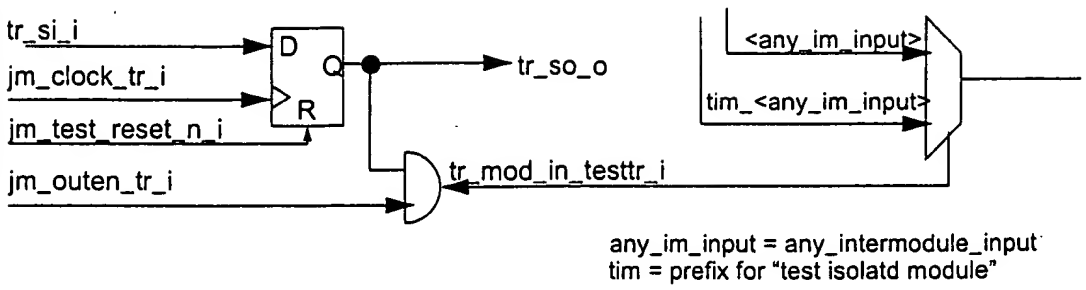
Save Results (Fault coverage,unvovered fauts)
Save constraints for the ATPG tool(set_test_hold, set_test_assume), if they have been used.

12.2. Preparation for isolated module test

The platform peripherals are tested in an isolated module test. The peripheral signals are made transparent at the chip boundary e.g. using multiplexers.
There are three categories of pins:
FPI bus: the FPI bus is made transparent via EBC/EBU
Alternate I/Os are made visible via normal prot functions
Outputs to other modules (here called intermoduleoutputs) are multiplexed in the ports
Inputs from other modules are multiplexed in the peripherals (intermoduleinputs)

12.2.1. Insert Testregister and Multiplex Inputs

Usually 1 test register bit is required. Use .... as an example
All InterModuleInputs are fed into a multiplexer and a testinput tim_.... is created, which will be connected to external pin on chip level



any_im_input = any_intermodule_input
tim = prefix for "test isolatd module"

tr_si_i, jm_clock_tr_si, jm_test_reset_n_i, jm_outen_tr_si are module inputs
tr_so_o is a module output. They will be connected at chip level

Further Cases:

- No test register: Some modules (SSC, ASC, IIC) have no dedicated intermodule inputs. In this cases the test register may be omitted. Such execptions have to be approved bye Georg Sigl, because they reduce testability.
- Addtional test register bits are necessary, if the module increases IDDQ-current (e.g. pullups, pulldowns) or the test register needs to be in a special mode for testing other peripherals (e.g. for clock generation)

12.2.2. Insert MISR

Insert Signature Registe Kann es je vorkommen, dass man das Signaturregister im Betrieb mitlaufen lassen moechte, dann braeuchten wir ein eigenes testregister Bit, ansosnter gemeinsam nit Eingansgsmultiplex

12.2.3. Insert MISR

You need a MISR, if your module has many inter module outputs (more than 5 to 10)

- Generate MISR using genbist (length is number of your intermodule outputs, but minimum 20
- add an asynchronous reset to the misr: jm_test_reset_i
- the bcode inputs of the MISR are test pins to your module: pdft_misr_bcode[1:0]
- the intermoduleoutputs are fed into the parallel MISR inputs
- the msb ofthe misr will be a test
- in test mode of the peripheral the MISR follows is controlled by the bcode test pins, in functional mode a synchronous reset is applied to bcode (bcode[1:0] ='10'
-

12.2.4. Select Module Pattern

The module pattern are used for a functional/performance test of the modul.
select a set of PDL files and store them.
Add a comment into your PDL file, where
If ATPG could not achieve 100% fault coverage make these

12.3. IDDQ Test preparation

IDDQ test vectors will be selected for platform products using Viewlogic/Sunrise. All peripheral have to fulfill the testability rules for this tool. Its not yet detemined, if we have to check every peripheral with Sunrise/testability checker.

12.4. Fault Coverage

12.5. Document Testability in Module Specification

Document Testability in your Module Specification. Use SSC as an example

13. Product Developers Test Cook Book

tbd

13.1. Preparations

13.1.1. Port Description

A description of the ports in computer readable format is necessary

13.2. Make Top Level Entity

13.2.1. Wiring of Test Register

13.2.2. Wiring of Scan Registers

13.3. Make Testbench

The necessay testbench contains